Enhancing Automata Learning by Log-Based Metrics

Petra van den Bos[⋆], Rick Smetsers[⋆⋆], and Frits Vaandrager[⋆⋆⋆]

Institute for Computing and Information Sciences, Radboud University, Nijmegen, the Netherlands

Abstract. We study a general class of distance metrics for deterministic Mealy machines. The metrics are induced by weight functions that specify the relative importance of input sequences. By choosing an appropriate weight function we may fine-tune a metric so that it captures some intuitive notion of quality. In particular, we present a metric that is based on the minimal number of inputs that must be provided to obtain a counterexample, starting from states that can be reached by a given set of logs. For any weight function, we may boost the performance of existing model learning algorithms by introducing an extra component, which we call the *Comparator*. Preliminary experiments show that use of the Comparator yields a significant reduction of the number of inputs required to learn correct models, compared to current state-of-the-art algorithms. In existing automata learning algorithms, the quality of subsequent hypotheses may decrease. Generalising a result of Smetsers et al., we show that the quality of hypotheses that are generated by the Comparator never decreases.

1 Introduction

In the platonic boolean world view of classical computer science, which goes back to McCarthy, Hoare, Dijkstra and others, programs can only be correct or incorrect. Henzinger [14] argues that this boolean classification falls short of the practical need to assess the behaviour of software in a more nuanced fashion against multiple criteria. He proposes to introduce quantitative fitness measures for programs, in order to measure properties such as functional correctness, performance and robustness. This paper introduces such quantitative fitness measures in the context of black-box testing, an area in which, as famously observed by Dijkstra [10], it is impossible to establish correctness of implementations.

The scenario that we consider in this paper starts from some legacy software component. Being able to retrieve models of such a component is potentially very useful. For instance, if the software is changed or enriched with new functionality, one may use a learned model for regression testing. Also, if the source code

⋆ Supported by STW project 13859 (SUMBAT).
⋆⋆ Supported by NWO project 628.001.009 (LEMMA).
⋆⋆⋆ Supported by STW project 11763 (ITALIA).

is hard to read and poorly documented, one may use a model of the software for model-based testing of a new implementation, or even for generating an implementation on a new platform automatically.

The construction of models from observations of component behaviour can be performed using model learning (e.g. regular inference) techniques [8]. One such technique is *active learning* [2, 25]. In active learning, a so-called Learner interacts with a System Under Learning (SUL), which is a black-box reactive system the Learner can provide inputs to and observe outputs from. By interacting with the SUL, the Learner infers a *hypothesis*, a state machine model that intends to describe the behaviour of the SUL. In order to find out whether a hypothesis is correct, we will typically use some conformance testing method. If the SUL passes the test, then the model is deemed correct. If the outputs of the SUL and the model differ, the test constitutes a counterexample, which may then be used by the Learner to construct an improved hypothesis.

Active learning has been successfully applied to learn models of (and find mistakes in) implementations of major protocols such as TCP [12] and TLS [9]. We have also used the approach to learn models of embedded control software at Océ [21] and to support refactoring of software at Philips HealthTech [19]. A key issue in black-box model learning, however, is assessing the correctness of the learned models. Since testing may fail to find a counterexample for a model, we can never be sure that a learned model is correct. Hence there is an urgent need for appropriate quantitative fitness measures.

Given a correct model S of the behaviour of the SUL, and a hypothesis model H, we are interested in distance metrics d that satisfy the following three criteria:

1. $d(H, S) = 0$ iff H and S have the same behaviour.
2. For any $\varepsilon > 0$, there exists a finite test suite T such that if H and S behave the same for all tests in T, it follows that $d(H, S) < \varepsilon$.
3. Metric d captures some intuitive notion of quality: the smaller $d(H, S)$, the better the quality of hypothesis H.

The first criterion is an obvious sanity property that any metric should satisfy. The second criterion says that, even though we can never exclude that H and S behave differently, we may, for any $\varepsilon > 0$, come up with a finite test suite T to check whether $d(H, S) < \varepsilon$. By running all the tests in T we either establish that H is a ε-approximation of S, or we find a counterexample that we can use to further improve our hypothesis model H. The third criterion is somewhat vague, but nevertheless extremely important. In practice, engineers will only be willing to invest further in testing if this leads to a quantifiable increase of demonstrated quality. They usually find it difficult to formalise their intuitive concept of quality, but typically require that a refactored implementation of a legacy component behaves the same for a set of common input sequences that have been recorded in log files, or specified as part of a regression test suite.

In this paper, we introduce a new, general class of metrics for deterministic Mealy machines that satisfy criteria (1) and (2). Our metrics are induced by weight functions that specify the relative importance of input sequences. By choosing an appropriate weight function we may fine-tune our metric so that it also meets criterion (3). In particular, we present metrics that are based on the minimal number of inputs that must be provided to obtain a counterexample starting from states that can be reached by a given set of logs. We also show that, given any weight function, we may boost the performance of existing learning algorithms by introducing an extra component, which we call the *Comparator*. Preliminary experiments show that use of the Comparator yields a significant reduction of the number of inputs required to learn a correct model for the SUL, compared to a current state-of-the-art algorithm. Existing learning algorithms do not ensure that the quality of subsequent hypotheses increases [23]. In fact, we may have $d(H', S) < d(H, S)$, even when hypothesis H is a refinement of hypothesis H'. Generalising a result of Smetsers et al. [23], we show that the quality of hypotheses never decreases when using the Comparator.

Related work Our research is most closely related to work of Smetsers et al. [23], which studies a simple distance metric, known from concurrency theory [3], in the setting of active learning. This metric is based on the minimal number of inputs required to obtain a counterexample: the longer this counterexample is, the closer a hypothesis is to the target model. Our work generalises the results of [23] to a much larger class of metrics, including log-based metrics that more accurately capture intuitive notions of quality.

The area of *software metrics* [24] aims to measure alternative implementations against different criteria. While software metrics mostly measure the quality of the software development process and static properties of code, our work is more ambitious since it considers the dynamic behaviour of software.

Henzinger [14] presents a general overview of work on behaviour-based metrics. Most research in this area thus far has been concerned with *directed metrics*, that is, metrics that are not required to be symmetrical. The idea is that for a given system X and requirement r, the distance function d describes the degree to which system X satisfies requirement r. Černỳ et al. [6], for instance, define a metric that is applied to an implementation and a specification. It is based on simulation relations between states of the two systems. If the specification simulates the implementation, then the distance is 0. However, if there is a state pair (q, q'), such that q does not simulate q', then a 'simulation failure game' is played. At a point where the specification has no transition with the same label as a transition of the implementation, the specification is allowed to choose some transition, at the cost of one penalty point. The distance between the two systems is then defined as the total number of penalty points reached when the implementation maximises, and the specification minimises the average number of penalty points. In our work we use metrics to compare hypotheses. Since we compare hypotheses in both directions, we use undirected (symmetric) metrics in our work. Thrane et al [26] study directed metrics between weighted automata [11]. In contrast, our work shows how weighted automata can be used to define undirected metrics between unweighted automata.

De Alfaro et al. [1] study directed and undirected metrics in both a linear time and a branching time setting. Most related to our work are the results on undirected linear distances. The starting point for the linear distances is the distance $\|\sigma - \rho\|_\infty$ between two traces σ and ρ, which measures the supremum of the differences in propositional valuations at corresponding positions of σ and ρ. The distance between two systems is then defined as the Hausdorff distance of their sets of traces. De Alfaro et al. [1] provide a logical characterization of these distances in terms of a quantitative version of LTL, and present algorithms for computing distances over metric transition systems, in particular an $\mathcal{O}(n^4)$ algorithm for computing distances between states of a deterministic metric transition system, where n is the size of the structure. The undirected linear distance metric of De Alfaro et al. [1] does not meet our second criterion for distance metrics (existence of finite test suites). However, the authors extend their results to a discounted context in which distances occurring after i steps are multiplied by α^i, where α is a discount factor in $[0, 1]$. We expect that finite test suites do exist for the discounted metrics of De Alfaro et al. [1], but it is not evident that the $\mathcal{O}(n^4)$ algorithm for computing distances generalizes to the metric setting.

There are intriguing relations between our results and the work of Brandán Briones et al [5] on a semantic framework for test coverage. In [5] also a general class of weight functions is introduced, which are called *weighted fault models*, which includes a finiteness condition in order to enable test coverage. However, weighted fault models do not induce a metric (since they may assign weight 0 to certain sequences) and hence the resulting theory is quite different.

2 Preliminaries

Sequences. Let I be any set. The set of finite sequences over I is denoted I^*. Concatenation of finite sequences is denoted by juxtaposition. We use ϵ to denote the empty sequence. The sequence containing a single element $e \in I$ is denoted as e. The length of a sequence $\sigma \in I^*$, i.e. the number of concatenated elements of σ, is denoted with $|\sigma|$. We write $\sigma \leq \rho$ to denote that σ is a prefix of ρ.

Mealy machines. We use Mealy machines as models for reactive systems.

Definition 1. *A Mealy machine is a tuple $M = (\Sigma, \Gamma, Q, q^0, \delta, \lambda)$, where Σ is a nonempty, finite set of inputs, Γ is a nonempty, finite set of outputs, Q is a finite set of states, $q^0 \in Q$ is the initial state, $\delta : Q \times \Sigma \to Q$ is a transition function, and $\lambda : Q \times \Sigma \to \Gamma$ is a transition output function. Functions δ and λ are extended to $Q \times \Sigma^*$ by defining, for all $q \in Q$, $i \in \Sigma$ and $\sigma \in \Sigma^*$,*

$$\delta(q, \epsilon) = q \;, \qquad \delta(q, i\sigma) = \delta(\delta(q, i), \sigma),$$
$$\lambda(q, \epsilon) = \epsilon \;, \qquad \lambda(q, i\sigma) = \lambda(q, i)\lambda(\delta(q, i), \sigma).$$

Observe that for each state and input pair exactly one transition is defined. The semantics of a Mealy machine are defined in terms of output functions:

Definition 2. *An output function over Σ and Γ is a function $A : \Sigma^* \to \Gamma^*$ that maps each sequence of inputs to a corresponding sequence of outputs such that, for all $\sigma, \rho \in \Sigma^*$, $|\sigma| = |A(\sigma)|$, and $\sigma \leq \rho \ \Rightarrow \ A(\sigma) \leq A(\rho)$.*

Definition 3. *The semantics of a Mealy machine M are defined by the output function A_M given by $A_M(\sigma) = \lambda(q^0, \sigma)$, for all $\sigma \in \Sigma^*$.*

Let $M_1 = (\Sigma, \Gamma, Q_1, q_1^0, \delta_1, \lambda_1)$ and $M_2 = (\Sigma, \Gamma, Q_2, q_2^0, \delta_2, \lambda_2)$ be two Mealy machines that share common sets of input and output symbols. We say M_1 and M_2 are *equivalent*, denoted $M_1 \approx M_2$, iff $A_{M_1} = A_{M_2}$. An input sequence $\sigma \in \Sigma^*$ *distinguishes* states $q_1 \in Q_1$ and $q_2 \in Q_2$ iff $\lambda_1(q_1, \sigma) \neq \lambda_2(q_2, \sigma)$. Similarly, σ distinguishes M_1 and M_2 iff $A_{M_1}(\sigma) \neq A_{M_2}(\sigma)$.

3 Weight functions and metrics

The metrics that we consider in this paper are parametrized by weight functions. Intuitively, a weight function specifies the importance of input sequences: the more weight an input sequence has, the more important it is that the output it generates is correct.

Definition 4. *A weight function for a nonempty, finite set of inputs Σ is a function $w : \Sigma^* \to \mathbb{R}^{>0}$ such that, for all $t > 0$, $\{\sigma \in \Sigma^* \mid w(\sigma) > t\}$ is finite.*

The finiteness condition in the above definition asserts that, even though the domain Σ^* is infinite, a weight function may only assign a value larger than t to a finite number of sequences, for any $t > 0$. Therefore, weight functions must involve some form of *discounting* by which long input sequences get smaller weights. This idea is based on the intuition that "a potential bug in the far-away future is less troubling than a potential bug today" [7].

Example 5. Let us define a weight function w by $w(\sigma) = 2^{-|\sigma|}$, for each $\sigma \in \Sigma^*$. Let $t \in \mathbb{R}^{>0}$. In order to see that w is a weight function, observe that

$$w(\sigma) > t \ \Leftrightarrow \ 2^{-|\sigma|} > t \ \Leftrightarrow \ |\sigma| < -\log_2 t.$$

Since Σ is finite, this implies that the set $\{\sigma \in \Sigma^* \mid w(\sigma) > t\}$ is finite, as required.

Below we define how a weight function induces a distance metric on output functions. Intuitively, the most important input sequence two output functions disagree on, i.e. the sequence with maximal weight, determines the distance.

Definition 6. *Let A, B be output functions over Σ and Γ, and let w be a weight function over Σ. Then the distance metric $d(A, B)$ induced by w is defined as:*

$$d(A, B) = \max\{w(\sigma) \mid \sigma \in \Sigma^* \wedge A(\sigma) \neq B(\sigma)\},$$

with the convention that $\max \emptyset = 0$. Sequence $\sigma \in \Sigma^*$ is a w-maximal distinguishing sequence for A and B if $A(\sigma) \neq B(\sigma)$ and $w(\sigma) = d(A, B)$.

Note that $d(A, B)$ is well-defined: the set $\{w(\sigma) \mid \sigma \in \Sigma^* \wedge A(\sigma) \neq B(\sigma)\}$ is either empty or contains, by the finiteness restriction for weight functions, a maximal element. Observe that for all output functions A and B with $A \neq B$ there exists a w-maximal distinguishing sequence.

Theorem 7. *Let Σ and Γ be nonempty, finite sets of inputs and outputs, and let $\mathcal{A}$ be the set of all output functions over Σ and Γ. Then the function d of Definition 6 is an ultrametric in the space $\mathcal{A}$ since, for any $A, B, C \in \mathcal{A}$,*

1. *$d(A, B) = 0 \Leftrightarrow A = B$ (identity of indiscernibles)*
2. *$d(A, B) = d(B, A)$ (symmetry)*
3. *$d(A, B) \leq \max(d(A, C), d(C, B))$ (strong triangle inequality)*

Proof.

1. If $A = B$ then $d(A, B) = 0$ by definition of d and the convention $\max \emptyset = 0$. We prove the converse implication by contraposition. Assume $A \neq B$. Then there exists a $\sigma \in \Sigma^*$ such that $A(\sigma) \neq B(\sigma)$. Since, by definition of w, $w(\sigma) > 0$ it follows, by definition of d, that $d(A, B) \neq 0$.
2. Follows directly from the symmetry in the definition of d.
3. We consider four cases:
   (a) If $d(A, B) = 0$ then $d(A, B) \leq \max(d(A, C), d(C, B))$ holds trivially.
   (b) If $d(A, C) = 0$ then $A = C$ by identity of indiscernibles and $d(A, B) \leq \max(d(A, C), d(C, B))$ holds trivially.
   (c) If $d(C, B) = 0$ then $C = B$ by identity of indiscernibles and $d(A, B) \leq \max(d(A, C), d(C, B))$ holds trivially.
   (d) Assume $d(A, B) \neq 0$, $d(A, C) \neq 0$ and $d(C, B) \neq 0$. Let σ_1 be a w-maximal distinguishing sequence for A and B, let σ_2 be a w-maximal distinguishing sequence for A and C, and let σ_3 be a w-maximal distinguishing sequence for C and B. Let $t_1 = w(\sigma_1)$, $t_2 = w(\sigma_2)$, and $t_3 = w(\sigma_3)$. We prove $t_1 \leq \max(t_2, t_3)$ by contradiction. Suppose $t_1 > \max(t_2, t_3)$. By definition of d, we know that for all σ with $w(\sigma) > t_2$, $A(\sigma) = C(\sigma)$. Similarly, we know that for all σ with $w(\sigma) > t_3$, $C(\sigma) = B(\sigma)$. Thus, for all σ with $w(\sigma) \geq t_1$, $A(\sigma) = C(\sigma) = B(\sigma)$. This contradicts the fact that $w(\sigma_1) = t_1$ and $A(\sigma_1) \neq B(\sigma_1)$.

For any weight function w, we lift the induced distance metric from output functions to Mealy machines by defining, for Mealy machines M and M', $d(M, M') = d(A_M, A_{M'})$. Observe that $d(M, M') = 0$ iff $M \approx M'$. Also, for each $\varepsilon > 0$, the set $\{\sigma \in \Sigma^* \mid w(\sigma) \geq \varepsilon\}$ is finite. Thus there exists a finite test suite that we may apply to either find a counterexample that proves $M \not\approx M'$ or to establish that $d(M, M') < \varepsilon$.

4 Log-based metrics

A *log* $\tau \in \Sigma^*$ is an input sequence that has been observed during execution of the SUL. We assume a finite set $L \subset \Sigma^*$ of logs that have been collected from the SUL. For technical reasons, we require that ϵ is included in L.

Let S be an unknown model of an SUL, and let H be a learned hypothesis for S. Since S and H are Mealy Machines, we may associate to each log $\tau \in L$ unique states $q \in Q_S$ and $q' \in Q_H$, that are reached by taking the transitions for the input symbols of τ, starting from q_S^0 and q_H^0 respectively. In this case, we say that τ *visits the state pair* (q, q'). Next, we can search for a sequence ρ that distinguishes q and q'. Now, $\tau\rho$ distinguishes q_S^0 and q_H^0, and hence $S \not\approx H$. We may define the distance of S and H in terms of the minimal number of inputs required to distinguish any pair of states (q, q') that is visited by some log $\tau \in L$.

We will now formalize the above intuition by defining a weight function and a distance metric. For this we need an auxiliary definition that describes how to decompose any trace σ into a maximal prefix that is contained in L, and a subsequent suffix.

Definition 8. *Let $\sigma \in \Sigma^*$ be an input sequence. An L-decomposition of σ is a pair (τ, ρ) such that $\tau \in L$ and $\tau\rho = \sigma$. We say that (τ, ρ) is a maximal L-decomposition if $|\tau|$ is maximal, i.e. for all L-decompositions (τ', ρ') of σ we have $|\tau'| \leq |\tau|$.*

Observe that, since $\epsilon \in L$, each sequence σ has a unique maximal L-decomposition (τ, ρ). We can now define the weight function w_L as a variant of the weight function of Example 5 in which the weight is not determined by the length of σ but rather by the length of the suffix ρ of the maximal L-decomposition.

Definition 9. *Let A be an output function over Σ and Γ, and let $\sigma \in \Sigma^*$. Then the weight function w_L is defined as $w_L(\sigma) = 2^{-|\rho|}$, where (τ, ρ) is the maximal L-decomposition of σ. We write d_L for the distance metric induced by w_L.*

In order to see that w_L is indeed a proper weight function in the sense of Definition 4, fix a $t > 0$ and derive:

$$\begin{aligned}
\{\sigma \in \Sigma^* \mid w_L(\sigma) > t\} &= \\
\{\sigma \in \Sigma^* \mid \exists \tau, \rho : 2^{-|\rho|} > t \wedge (\tau, \rho) \text{ is a maximal } L\text{-decomposition of } \sigma\} &\subseteq \\
\{\tau\rho \in \Sigma^* \mid 2^{-|\rho|} > t \wedge \tau \in L\} &= \\
\{\tau\rho \in \Sigma^* \mid \ |\rho| < -\log_2 t \wedge \tau \in L\}&
\end{aligned}$$

Since both Σ and L are finite the last set is finite, and therefore the first set is finite as well.

Observe that the metric d_L coincides with the metric of [3, 23] if we take as set L of logs the singleton set $\{\epsilon\}$, as we then only take into account w-maximal distinguishing sequences starting in the initial state.

Example 10. Let us illustrate our log-based metric with a simple coffee machine. The machine is always used as follows. First, a coffee pod is placed, then the machine is provided with water, then the button is pressed to obtain coffee, and finally the machine is cleaned. The logs for the coffee machine consist of the sequence *pod water button clean* and all of its proper prefixes (i.e., *pod water button*, *pod water*, *pod*, and the empty sequence ϵ).

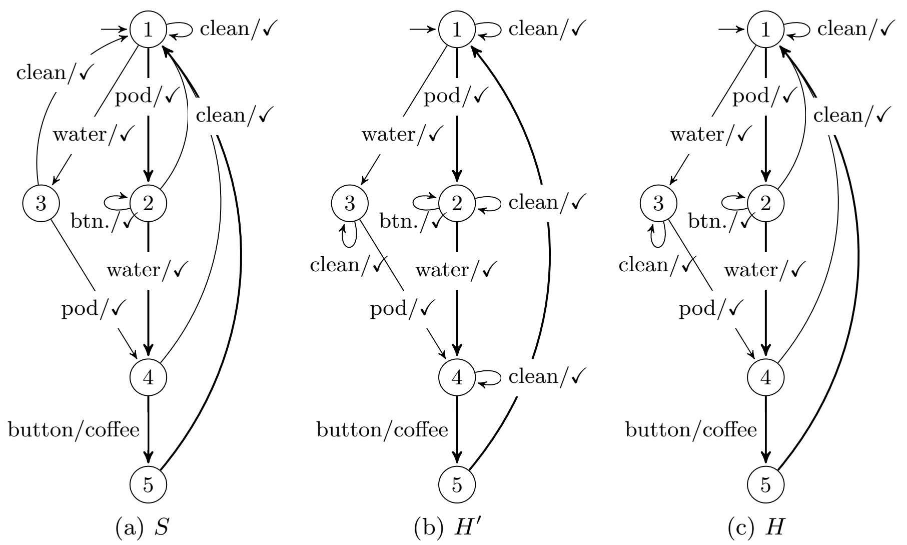

Fig. 1. Models of a coffee machine. The machine has one button (abbreviated as btn. in state 2), can be provided with a pod and water, and can be cleaned. It can produce coffee, or remain quiescent (✓) after an input. The logged trace is displayed with bold arrows. Some insignificant self-loops are not displayed.

Figure 1 presents three models for the coffee machine. The model shown in Figure 1a is the correct model S, and the models shown in Figure 1b and 1c, respectively, are hypotheses H' and H for S. Observe that both hypotheses produce correct output for all logs, but that they nevertheless have some incorrect transitions. In H', the *clean* transitions are incorrect in states 2, 3 and 4. In H only the erroneous *clean* transition in state 3 remains.

Let us compute the distances of H' and H to S. A w_L-maximal distinguishing sequence to discover inequivalence of H' and S is *pod water clean button*. At the end of this sequence, H' outputs *coffee*, while S remains quiescent, i.e., output ✓. Despite that the sequence is of length four, it only takes two inputs to discover the error starting from a state that can be reached via a log, since state 4 is reached by *pod water*. Therefore, the distance between H' and S according to our metric is 2^{-2}.

A w_L-maximal distinguishing sequence to discover the remaining error in H is *water clean pod button*: H outputs *coffee* at the end of this sequence, where it should remain quiescent. Since the prefix *water* has never been observed in logs, it takes four inputs to discover this error starting from a state that has been visited by a log: state 1 is known because it is reached by the empty sequence ϵ. As a result, the distance between H and S according to our metric is 2^{-4}.

Observe that these distances capture the subtle improvement in H compared to H' (as $2^{-4} < 2^{-2}$), despite that four inputs are required in both hypotheses to discover an error. Both H' and H are wrong, but the problem with H' is more serious, as the error is visible after two transitions starting from a state that is reached during normal use of the system, instead of four transitions in H. In the metric of [3, 23], both hypotheses would be considered equally distant to S for this reason.

Algorithm. Van den Bos [4] presents an algorithm for finding w_L-maximal distinguishing sequences for two given models. As we will see, such an algorithm is extremely useful as a component in model learning. The input of the algorithm of Van den Bos [4] consists of two Mealy machines H and H' that agree on all inputs from L, that is, $A_H(\sigma) = A_{H'}(\sigma)$, for all $\sigma \in L$. (This can be realized, for instance, by first checking for each hypothesis model whether it is consistent with all the logs in L.) The key idea of the algorithm is that minimal length distinguishing sequences (for pairs of states) are gathered by constructing a partition of indistinguishable states. By processing the partition, a distinguishing sequence of minimal length is found for each pair of states in $Q_H \times Q_{H'}$, or it is established that the states are equivalent. After that, a w_L-maximal distinguishing sequence can be found by picking a minimal length distinguishing sequence that is visited by some log in L. Intuitively, the time complexity of the search for these sequences can be deferred from the fact that a table has to be filled for all state pairs. Indeed, it follows that the algorithm is quadratic, i.e. of $\mathcal{O}(pn^2)$, where n is the sum of the number of states of H and H', and p is the number of inputs. In [22] it is shown that minimal length distinguishing sequences for all pairs of states can even be found in $\mathcal{O}(pn \log n)$.

Weighted automata. There are many possible variations of our log-based metrics. We may for instance consider variations in which the weight of a log is partially determined by its frequency. We may also assign a higher weight to logs in which certain "important" inputs occur. All such variations can be easily defined using the concept of a *weighted automaton* [11], i.e., an automaton in which states and transition carry a certain weight. Below we define a slightly restricted type of weighted automaton, called *weighted Mealy machine*, which only assigns weights to transitions.

Definition 11. *A weighted Mealy machine is a tuple $M = (\Sigma, \Gamma, Q, q^0, \delta, \lambda, c)$, where $(\Sigma, \Gamma, Q, q^0, \delta, \lambda)$ is a Mealy machine and $c : Q \times \Sigma \to \mathbb{R}^{>0}$ is a cost function. Cost function c is extended to $Q \times \Sigma^*$ by defining, for all $q \in Q$, $i \in \Sigma$ and $\sigma \in \Sigma^*$, $c(q, \epsilon) = 1$ and $c(q, i\sigma) = c(q, i) \cdot c(\delta(q, i), \sigma)$. The cost function $c_M : \Sigma^* \to \mathbb{R}^{>0}$ induced by M is defined as $c_M(\sigma) = c(q^0, \sigma)$.*

A cost function c_M is not always a weight function in the sense of Definition 4, since it may assign an unbounded weight to infinitely many sequences. However, if the weight of any *cycle* in M is less than 1 then c_M is a weight function.

Definition 12. *Let $M = (\Sigma, \Gamma, Q, q^0, \delta, \lambda, c)$ be a weighted Mealy machine. A path of M is an alternating sequence $\pi = q_0 i_0 q_1 \cdots q_{n-1} i_{n-1} q_n$ of states in Q and inputs in Σ, beginning and ending with a state, such that, for all $0 \leq j < n$, $\delta(q_j, i_j) = q_{j+1}$. Path π is a cycle if $q_0 = q_n$ and $n > 0$. The weight of path π is defined as the product of the weights of the contained transitions: $\prod_{j=0}^{n-1} c(q_j, i_j)$.*

Theorem 13. *Let M be a weighted Mealy machine, then c_M is a weight function iff all cycles have weight (strictly) less than 1.*

Let L be a prefix closed set of logs (i.e. all prefixes of a log in L are also included in L). Then the weight function w_L of Definition 9 can alternatively be defined as the weight function c_M induced by a weighted automaton M with states taken from $L \cup \{\perp\}$, that is, the set of logs extended with an extra sink state $\perp$, initial state ϵ, and transition function δ and cost function c defined as:

$$\delta(q,i) = \begin{cases} qi & \text{if } q \in L \wedge qi \in L \\ \perp & \text{otherwise} \end{cases} \qquad c(q,i) = \begin{cases} 1 & \text{if } q \in L \wedge qi \in L \\ \frac{1}{2} & \text{otherwise} \end{cases}$$

Note that, by Theorem 13, c_M is indeed a weight function.

5 An adapted learning algorithm

In this section, we will explain how weight functions and their induced metrics can be used to improve model learning.

Active learning is a learning framework in which a Learner can ask questions (*queries*) to a Teacher, as visualized in Figure 2a. We assume that the Teacher is capable of answering queries correctly according to the Minimally Adequate Teacher (MAT) model of Angluin [2]. The Teacher knows a Mealy machine S which is unknown to the Learner. Initially, the Learner only knows the input and output symbols of S. The task of the Learner is to learn S by asking two types of queries:

- With a *membership query* (MQ), the Learner asks what the response is to an input sequence $\sigma \in \Sigma^*$. The Teacher answers with the output sequence $A_S(\sigma)$.
- With an *equivalence query* (EQ), the Learner asks whether a hypothesized Mealy machine H is correct, that is, whether $H \approx S$. The Teacher answers *yes* if this is the case. Otherwise it answers *no* and supplies a *counterexample*, which is a sequence $\sigma \in \Sigma^*$ that produces a different output sequence for both Mealy machines, that is, $A_H(\sigma) \neq A_S(\sigma)$.

Starting from Angluin's seminal L^* algorithm [2], many algorithms have been proposed for learning a Mealy machine H that is equivalent to S via a finite number of queries. We refer to [15] for an excellent recent overview. In applications in which one wants to learn a model of a black-box reactive system, the Teacher typically consists of a System Under Learning (SUL) that answers the membership queries, and a conformance testing (CT) tool [16] that approximates the equivalence queries using a set of *test queries* (TQs). A test query consists of asking the SUL what the response is to an input sequence $\sigma \in \Sigma^*$, similar to a membership query. A schematic overview of such an implementation of active learning is shown in Figure 2b.

We will now explain how weight functions and the metrics they induce can be used to enhance active learning. Our idea is to place a new "Comparator" component in between the Learner and the Teacher, as displayed in Figure 3. The Comparator ensures that the distance of subsequent hypotheses to the target model S never increases. Moreover, the Comparator may replace an equivalence query by a single membership query. This speeds up the learning process, since a Teacher typically answers an equivalence query by running a large number of test queries generated by a conformance testing algorithm. In the printer controller case study of [21], for instance, on average more than 270.000 test queries were used to implement a single equivalence query.

Assume we have a weight function w and an oracle which, for given models H and H' with $H \not\approx H'$, produces a w-maximal distinguishing sequence, that is, a sequence $\sigma \in \Sigma^*$ with $d(H, H') = w(\sigma)$ and $A_H(\sigma) \neq A_{H'}(\sigma)$. The behavior of the Comparator component can now be described as follows:

- The first equivalence query from the Learner is forwarded to the Teacher, and the resulting reply from the Teacher is forwarded again to the Learner.
- The Comparator always remembers the last hypothesis that it has forwarded to the Teacher.
- Upon receiving any subsequent equivalence query from the Learner for the current hypothesis H, the Comparator computes a w-maximal distinguishing sequence σ for H and the previous hypothesis H', as described in the previous section. The Comparator poses a membership query σ to the Teacher and awaits the reply $A_S(\sigma)$. Depending on the reply, two things may happen:
  1. $A_S(\sigma) \neq A_H(\sigma)$. The Comparator has found a counterexamples for hypothesis H, and returns *no* together with σ to the Learner in response to the equivalence query.
  2. $A_S(\sigma) = A_H(\sigma)$. The Comparator forwards the equivalence query to the Teacher, waits for the reply, and forwards this to the Learner.

From the perspective of the Learner, the combination of a Comparator and a Teacher behave like a regular Teacher, since all membership and equivalence queries are answered appropriately and correctly. Hence the Learner will succeed to learn a correct hypothesis H after posing a finite number of queries.

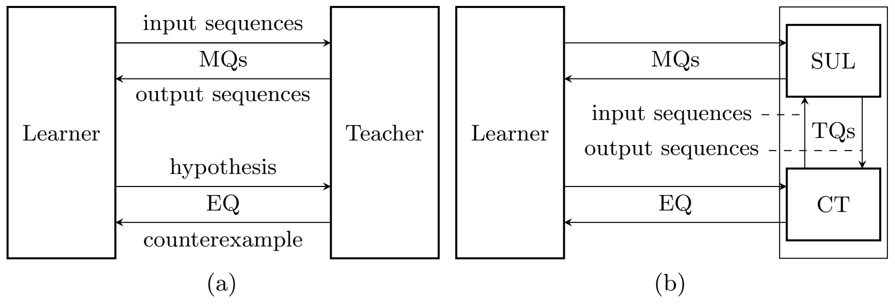

Fig. 2. Active learning framework (a) and implementation for black-box learning (b).

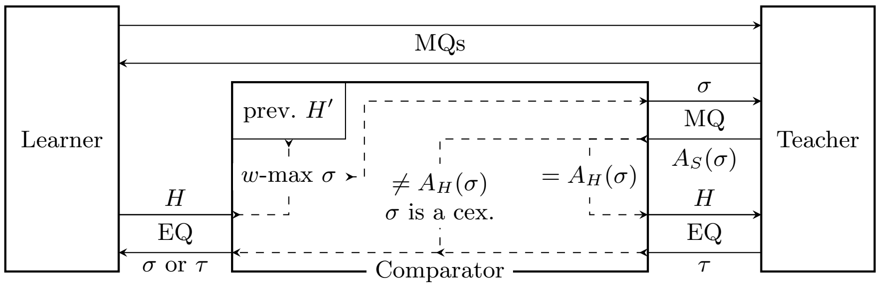

Fig. 3. Active learning framework with the Comparator in the middle.

Conversely, from the perspective of the Teacher, the Comparator and the Learner together behave just like a regular Learner that poses membership and equivalence queries. A key property of the Comparator/Learner combination, however, is that the quality of hypotheses never decreases. We claim that, whenever the Comparator first poses an equivalence query for H' and then for H, we always have $d(S, H) \leq d(S, H')$. In order to see why this is true, observe that when the Comparator poses the equivalence query for H it has found a w-maximal distinguishing sequence σ for H and H'. Therefore we know that $A_H(\sigma) \neq A_{H'}(\sigma)$ and

$$w(\sigma) = d(H', H) \tag{1}$$

Through a membership query σ the Comparator has also discovered that $A_S(\sigma) = A_H(\sigma)$. This implies $A_S(\sigma) \neq A_{H'}(\sigma)$ and thus

$$w(\sigma) \leq d(S, H') \tag{2}$$

Now we infer

$$\begin{aligned} d(S,H) &\leq \text{(by the strong triangle inequality, Theorem 7)} \\ \max(d(S,H'), d(H',H)) &= \text{(by equation (1))} \\ \max(d(S,H'), w(\sigma)) &= \text{(by inequality (2))} \\ d(S,H'). & \end{aligned}$$

Hence, the distance between subsequent hypotheses and S never increases.

6 Case studies

In this section, we present two case studies in which we measure the effect of a Comparator for the log-based metrics from Section 4. In the first case study, we learn a model for the Engine Status Manager (ESM), a piece of industrial software that controls the transition from one status to another in Océ printers. In the second case study we learn a model for the Windows 8 TCP server.

Table 1. Number of inputs used to learn a model for the ESM ($n = 20$).

setup	mean	std. dev.	median	min	max
(a)	416 519 487	119 015 166	404 307 465	109 781 273	686 385 316
(b)	371 248 375	57 005 155	377 724 597	290 072 340	545 535 231
(c)	308 928 853	50 719 369	295 863 646	243 197 179	430 523 416

Engine Status Manager. In [20], a first attempt was made to learn a model for the ESM using the algorithm of Rivest & Schapire [18] as implemented in LearnLib [17]. A manually constructed reference model was used to determine the success of the learning algorithm. The author did not succeed in learning the complete model, as it took the Teacher too long to find a counterexample at some point. A second attempt was made in [21]. In this work, an adaptation of a finite state machine testing algorithm by Lee and Yannakakis [16] was used by the Teacher to find counterexamples. The authors succeeded in learning a complete, correct model with 3410 states for the ESM through a sequence of more than 100 hypotheses. Particularly because of the large number of hypotheses, this case study appeared to be a suitable case to test the impact of a Comparator.

Using the same setup for the Learner and the Teacher as in [21], we have conducted twenty independent runs for each of the following three experiments.

(a) The classical setting without a Comparator.
(b) A setting with a Comparator and the trivial log set $L = \{\epsilon\}$. This setup resembles the algorithm presented by Smetsers et al. [23].
(c) A setting with a Comparator, using the aforementioned algorithm for finding w-maximal distinguishing sequences on a nontrivial set of logs.

No real logs were available to us for setup (c), because no appropriate logging method was in place to obtain real user logs from, and setting up such logging would be tedious. Instead, we developed a method to generate logs that resemble real logs. In [20], a couple of 'paths', directly inferred from the ESM, are given. Such a path is a sequence of subsets of the input alphabet. An input sequence for the ESM can be obtained by concatenating inputs from the subsequent subsets of the path. More specifically, the algorithm for doing this keeps track of the sequence σ it has constructed, the current state q, the set of already visited states V, and the index k of the current subset of the path. Initially, $\sigma = \epsilon$, $q = q^0$, $V = \{q^0\}$, and $k = 0$. The algorithm extends σ with an input i from subset k if $\delta(q, i) \notin V$. In that case, q and V are updated accordingly. Else, we search for an input in subset $k + 1$. Only sequences with their last input in the last subset of a path are included in the logs. In total, we have generated 9800 logs for each run.

Experimental results are shown in Table 1. On average over 20 runs, setup (c) (with Comparator) requires 25.8% fewer inputs than setup (a) (no Comparator), and 16.8% fewer inputs than setup (b) (the algorithm of [23]) to learn a correct model for the ESM. A non-parametrical, distribution independent statistical test was used to determine that this result is significant ($p < 0.05$, $z = -4.15$).

Table 2. Number of inputs used to learn a model for a TCP server ($n = 500$).

setup	mean	std. dev.	median	min	max
(a)	163 463	154 353	106 750	35 694	1 076 538
(b)	162 948	191 222	105 487	40 927	2 380 343
(c)	159 409	141 255	110 545	41 471	1 168 348

TCP. In [13], active learning was used to obtain a model for the Windows 8 TCP server. Using the aforementioned Learner and Teacher algorithms, the authors succeeded in learning a model of 38 states through a series of 13 hypotheses. We have conducted 500 runs for each of the experimental setups described above, using the model of [13] as an SUL. Experimental results are shown in Table 2. Unfortunately, we found no significant reduction in inputs when using the Comparator. We conjecture that this is due to the inherent simplicity of the model.

7 Conclusions and future work

We have presented a general class of distance metrics on Mealy machines that may be used to formalize intuitive notions of quality. Preliminary experiments show that our metrics can be used to obtain a significant reduction of the number of inputs required to learn large black-box models. For smaller models, no reduction was found. Therefore, we conjecture that the utility of our metrics increases as models become more complex. In future work, we plan to perform more experiments to verify these results. In addition, we wish to do experiments with real logs, instead of generated ones. Another topic for future research is to develop efficient algorithms for computing w-maximal distinguishing sequences for the weight functions induced by weighted Mealy machines. Bounding the distance between a hypothesis and the unknown target model during learning remains a challenging problem. Our experiments have produced discouraging results in this sense, since the quality of a hypothesis is hard to predict because of the high variance for different experimental runs.

References

1. L. De Alfaro, M. Faella, and M. Stoelinga. Linear and branching system metrics. *Software Engineering, IEEE Transactions on*, 35(2):258–273, 2009.
2. D. Angluin. Learning regular sets from queries and counterexamples. *Information and Computation*, 75(2):87–106, 1987.
3. J.W. De Bakker and J.I. Zucker. Processes and the denotational semantics of concurrency. *Information and Control*, 54(12):70–120, 1982.
4. Petra Van den Bos. Enhancing active automata learning by a user log based metric. Master's thesis, Radboud University Nijmegen, 2015.
5. L. Brandán Briones, E. Brinksma, and M. Stoelinga. A semantic framework for test coverage. In S. Graf and W. Zhang, editors, *Proc. ATVA*, volume 4218 of *LNCS*, pages 399–414. Springer Berlin Heidelberg, 2006.

6. P. Černý, Th. Henzinger, and A. Radhakrishna. Simulation distances. In P Gastin and F. Laroussinie, editors, *Proc. CONCUR*, pages 253–268. Springer Berlin Heidelberg, 2010.
7. L. De Alfaro, Th. Henzinger, and R. Majumdar. Discounting the future in systems theory. In J. C. M. Baeten, J. K. Lenstra, J. Parrow, and G. J. Woeginger, editors, *Proc. ICALP*, pages 1022–1037. Springer Berlin Heidelberg, 2003.
8. C. de la Higuera. *Grammatical inference*. Cambridge University Press, Cambridge, UK, 2010.
9. J. de Ruiter and E. Poll. Protocol state fuzzing of tls implementations. In *USENIX Security 15*, pages 193–206, Washington, D.C., August 2015. USENIX Association.
10. E. W. Dijkstra. The humble programmer. *CACM*, 15(10):859–866, 1972.
11. M. Droste, W. Kuich, and H. Vogler. *Handbook of Weighted Automata*. Springer Publishing Company, Incorporated, 1st edition, 2009.
12. P. Fiterău-Broştean, R. Janssen, and F. Vaandrager. Learning fragments of the TCP network protocol. In F. Lang and F. Flammini, editors, *Proc. FMICS*, volume 8718 of *LNCS*, pages 78–93. Springer International Publishing, 2014.
13. P. Fiterău-Broştean, R. Janssen, and F. Vaandrager. Combining model learning and model checking to analyze TCP implementations, 2016. Submitted to CAV.
14. Th. Henzinger. Quantitative reactive modeling and verification. *Computer Science - Research and Development*, 28(4):331–344, 2013.
15. M. Isberner. *Foundations of Active Automata Learning: An Algorithmic Perspective*. PhD thesis, Technical University of Dortmund, 2015.
16. D. Lee and M. Yannakakis. Principles and methods of testing finite state machines-a survey. *Proc. of the IEEE*, 84(8):1090–1123, 1996.
17. H. Raffelt, B. Steffen, T. Berg, and T. Margaria. LearnLib: a framework for extrapolating behavioral models. *STTT*, 11(5):393–407, 2009.
18. R.L. Rivest and R.E. Schapire. Inference of finite automata using homing sequences. *Information and Computation*, 103(2):299–347, 1993.
19. M. Schuts, J. Hooman, and F. Vaandrager. Refactoring of legacy software using model learning and equivalence checking: an industrial experience report. In *Proc. iFM*, 2016.
20. W. Smeenk. Applying automata learning to complex industrial software. Master's thesis, Radboud University Nijmegen, September 2012.
21. W. Smeenk, J. Moerman, F. Vaandrager, and D.N. Jansen. Applying automata learning to embedded control software. In M. Butler, S. Conchon, and F. Zaïdi, editors, *Proc. ICFEM*, volume 9407 of *LNCS*, pages 67–83. Springer International Publishing, 2015.
22. R. Smetsers, J. Moerman, and D.N. Jansen. Minimal separating sequences for all pairs of states. In *Proc. LATA*, 2016.
23. R. Smetsers, M. Volpato, F. Vaandrager, and S. Verwer. Bigger is not always better: on the quality of hypotheses in active automata learning. In *JMLR: W&CP 34; Proceedings of ICGI*, pages 167–181, 2014.
24. I. Sommerville. *Software engineering*. Addison-Wesley Publishing Company, 2001.
25. Bernhard Steffen, Falk Howar, and Maik Merten. Introduction to active automata learning from a practical perspective. In *Proc. SFM*, volume 6659 of *LNCS*, pages 256–296. Springer Berlin Heidelberg, 2011.
26. C. Thrane, U. Fahrenberg, and K.G. Larsen. Quantitative analysis of weighted transition systems. *Journal of Logic and Algebraic Programming*, 79(7):689–703, 2010.